(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 887 751 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
30.12.1998 Bulletin 1998/53

(51) Int Cl.[6]: **G06F 17/30**

(21) Application number: **98305007.1**

(22) Date of filing: **25.06.1998**

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **27.06.1997 US 884252**

(71) Applicant: **Sun Microsystems, Inc.**
Palo Alto, California 94303-4900 (US)

(72) Inventors:
- **Cramer, Samuel M.**
  Mountain View, California 94040 (US)
- **Skinner, Glenn C.**
  Palo Alto, California 94306 (US)

(74) Representative: **Harris, Ian Richard et al**
D. Young & Co.,
21 New Fetter Lane
London EC4A 1DA (GB)

(54) Method and system for maintaining a global name space

(57) A global mount mechanism capable of maintaining a consistent global name space in a distributed computing system including a plurality of nodes interconnected by a communications link is herein disclosed. The global mount mechanism mounts a new file system resource into the global name space in a coherent manner such that the new file system resource is mounted at the same mount point concurrently in each node. The global mount mechanism accommodates mount or unmount requests initiated from a requesting node for a resource located in a remote node. The global mount mechanism is also used to unmount a file system resource from the global name space. The global mount mechanism also includes an initialization procedure that is used to generate the global name space initially by providing each local mount point with a global locking capability.

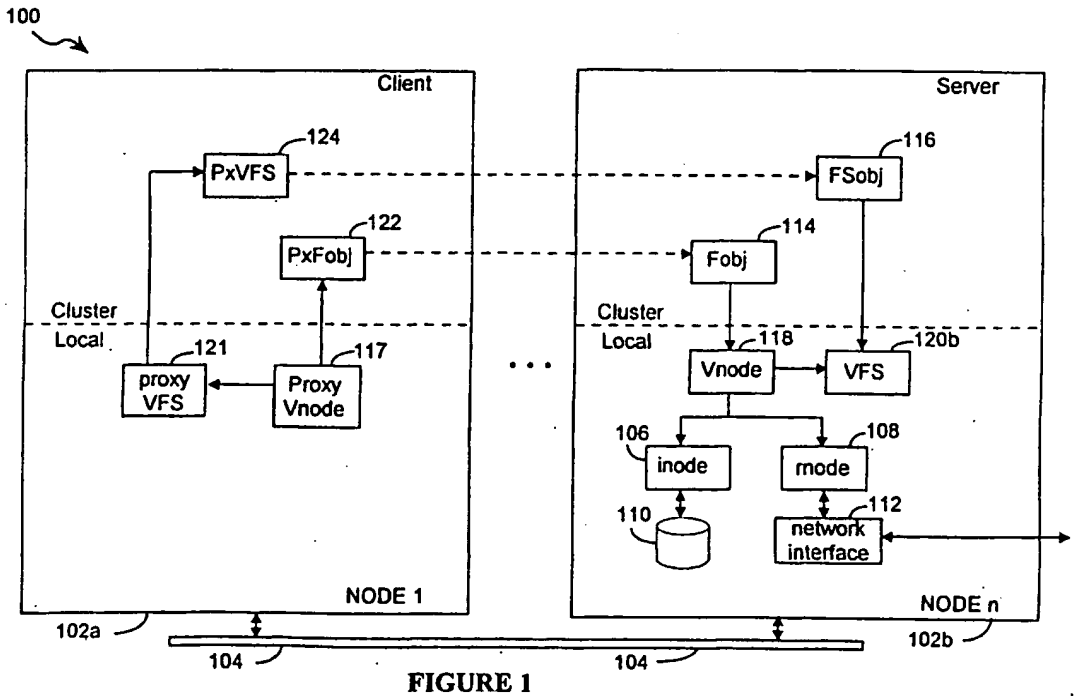

FIGURE 1

EP 0 887 751 A2

Description

The present invention relates generally to a distributed file system and particularly to a method and system for maintaining a global name space in a distributed file system.

BACKGROUND OF THE INVENTION

A cluster is a group of independent computing nodes connected by a high-speed communications link. Each computing node has one or more processes where each process has its own address space. Each process can access data that is associated with a file system that exists in the cluster. The file system can be resident in the node associated with the process or in another node within the cluster.

The cluster has a global name space which represents the file systems accessible to each node within the cluster. Each node may also have a local name space representing the file systems accessible to processes associated with a particular node. A user associated with a particular node can mount or connect a file system local to one node into the global name space. Furthermore, a user can unmount or disconnect a file system from the global name space thereby making the file system inaccessible to each node in the cluster.

It is beneficial for each node to have a single system image of the global name space. However, maintaining this image is complicated by issues of coherency, resource location, and transparency. Coherency must be achieved in mounting and unmounting a file system at the same mount point within the cluster and at the same point in time. Otherwise, each node can mount a file system at a different mount point or access an unmounted file.

From the view point of users issuing mount and unmount commands, the existence of the global name space should be as transparent as possible. This transparency will minimize the changes required to the interface of the mount and unmount command as well as to user application programs and data.

Furthermore, in some instances the resources needed to mount a file system are not always accessible from all nodes in the cluster. This can affect the mount of a file system initiated from one node when the resources associated with the file system are best accessed from another node. In order to perform the mount task, it becomes necessary to overcome this obstacle.

Accordingly, there exists a need to maintain a global name space in a distributed computing environment in a manner that accounts for the aforementioned constraints.

SUMMARY OF THE INVENTION

Particular and preferred aspects of the invention are set out in the accompanying independent and dependent claims. Features of the dependent claims may be combined with those of the independent claims as appropriate and in combinations other than those explicitly set out in the claims.

An embodiment of the present invention provides a global mount mechanism capable of maintaining a consistent global name space in a distributed computing system. The distributed computing system includes a cluster of nodes interconnected by a communications link. The global mount mechanism mounts a new file system resource into the global name space and unmounts a mounted file system resource in a coherent manner. Coherency is achieved by mounting the file system resource at the same mount point within the cluster and at the same point in time. The global mount mechanism utilizes a distributed locking mechanism to ensure that the mount or unmount operation is performed in a coherent manner. The global mount mechanism accounts for the disparity in file system resource distribution by allowing a file system resource to be mounted by a node not associated with the file system resource .

The global name space is a collection of file system resources that are accessible from each node in the cluster. Each file system resource mediates access to a set of file resources belonging to its associated file system resource. Each file resource is represented by a pathname that can include one or more directories. Each directory in the global name space can serve as a global mount point at which a new file system resource can be mounted or incorporated into the global name space. When a new file system resource is mounted at a particular mount point, the file resources it mediates become accessible through pathnames that start with the mount point's pathname.

A server node that is associated with a file system resource includes a virtual file system (VFS) mechanism and a file system object (FSobj) to represent the file system resource. In addition, each client node includes a proxy VFS mechanism and proxy FSobj to represent the file system resource.

A virtual file system node (vnode) mechanism is used to represent each directory in the global name space. The vnode mechanism is used as a mount point at which a VFS or proxy VFS mechanism is attached thereby incorporating the new file system resource into the global name space.

The global mount mechanism includes an initialization mechanism that generates the global name space initially. At system initialization, each node has a local name space including a number of local file system resources that are only accessible from within the node. The initialization mechanism gives one or more local directories or local mount points a global locking capability that enables the local mount to be locked by any node in the cluster. The global locking capability turns the local mount point into a global mount point that is part of

the global name space. One or more local file system resources can then be mounted at a global mount point and, hence, become part of the global name space.

BRIEF DESCRIPTION OF THE DRAWINGS

Exemplary embodiments of the invention are described hereinafter, by way of example only, with reference to the accompanying drawings, in which:

Fig. 1 is a block diagram of a distributed computing system incorporating the preferred embodiments of the present invention.

Fig. 2A represents an exemplary global name space and a file system that will be mounted into the global name space.

Fig. 2B represents the global name space of Fig. 2A after the mount of the file system is performed.

Fig. 3A represents an exemplary distributed file system of the global name space shown in Fig. 2A.

Fig. 3B represents an exemplary distributed file system of the global name space shown in Fig. 2B.

Fig. 4 is a block diagram of a distributed computing system incorporating the preferred embodiments of the present invention.

Figs. 5A and 5B illustrate the vnode and VFS data structures used in an embodiment of the present invention.

Fig. 5C illustrates the PxFobj data structure used in an embodiment of the present invention.

Fig. 6 is a flow chart illustrating the steps used by the global mount mechanism in mounting a new file system resource into the global name space.

Fig. 7 illustrates the distributed locking mechanism used in an embodiment of the present invention.

Figs. 8A - 8D illustrate by way of an example the global mount mechanism of Fig. 6.

Fig. 9 illustrates the steps used by the global mount mechanism in unmounting a mounted file system resource from the global name space.

Fig. 10 illustrates by way of an example the global unmount mechanism of Fig. 9.

Fig. 11 illustrates the steps used to generate a global mount point in a local name space.

Fig. 12 illustrates the distributed locking mechanism used in Fig. 11.

DESCRIPTION OF THE PREFERRED EMBODIMENT

Overview of the File System Data Structures

Referring to Fig. 1, there is shown a distributed computing system 100 including a plurality of computing nodes 102. Each computing node 102 represents an independent client/server computer that is interconnected via a communications link 104. Each node can act as either a client or a server computer or both. With respect to a given file system resource, one node can act as the server computer for the resource and other nodes as client computers. A client computer is associated with a node that accesses file system resources over the communication link and a server computer is associated with a node that provides file system resources over the communication link. However, the classification of a client and server computer for a particular file system resource can vary over time.

The distributed computing system 100 utilizes a distributed file system that includes a local layer (Local) and a cluster layer (Cluster). The local layer includes a physical file system and a vnode/VFS interface. The physical file system is any file system that stores file data on a data storage device that is local to the node. Examples of physical file systems can include but are not limited to the MSDOS PC file system, the 4.3BSD file system, the Sun network file system (NFS), and the like.

The vnodeNFS interface is an interface between the operating system and the physical file system. The vnode/VFS interface accommodates multiple file system implementations within any Unix operating system or kernel. A file system can be incorporated into the kernel through the vnode/VFS interface. A vnode (i.e., virtual file node) 118 is a data structure that contains operating system data describing a particular file. A virtual file system (VFS) 120 is a data structure that contains operating system data describing a particular file system.

The cluster layer represents the file system resources that are accessible from any node within the cluster. It should be noted that the term "file system resource " as used herein represents information need to characterize a set of files, a file system, a directory or a group of directories. In addition, a directory can be considered a file. In the cluster file layer, each file system resource is represented as an object. A server node 102b will have a file system object (FSobj) 116 for each file system resource or file system under its control and a file object (Fobj) 114 for each directory that is under the server node's control. A client node 102a will have a proxy file object (PxFobj) 122 for each file that is accessed from a remote node and a proxy file system object (PxVFS) 124 for each file system or resource that is accessed from a remote node. In the case where the

client and server computer are the same node, the client node 102a will have a file system object (FSobj) 116 and a file object (Fobj) 114 for a file system resource when the client node 102a is acting as the server for the file system resource and will have proxy data structures when the client node 102a is acting as a client for the resource. The proxy file object 122 contains an object reference to the associated file object 114 and the proxy file system object 122 contains an object reference to the associated file system object 116.

The client and server communicate through remote procedure calls (RPCs). One or more threads associated with a client node can access a remote file system resource through a remote object invocation using the proxy object reference in a RPC. A more detailed description pertaining to the implementation of the remote object invocation can be found in pending U.S. Patent Application, serial no. _____ entitled "A System and Method for Remote Object Invocation," filed June 19,1997, and assigned to Sun Microsystems Inc.

The cluster layer represents the global name space. In addition, each node has a local name space representing file systems or resources that are locally accessible only to that node. A node can incorporate one or more file systems into the global name space with the mount command and can remove file systems from the global name space with the unmount command. A more detailed description of the vnode and VFS interfaces can be found in Kleiman, Steven R., "Vnodes: An Architecture for Multiple File System Types in Sun UNIX," Proceedings of the Summer 1986 USENIX Conference, Atlanta, 1986.

Fig. 1 illustrates the aforementioned infrastructure of the local and cluster layers for an exemplary file system containing a root directory and the file *myfile.c* as shown below.

/ (root directory) *myfile.c*

The local layer of the distributed file system on the server node 102b includes a vnode 118 representing the file *myfile.c*. Each vnode contains a reference to data that is specific to the kind of file system that it represents (i.e., file system specific data). A vnode contains one such reference. These references can vary. In Fig. 1, there are shown two such references 106,108 and it should be noted that the two references are shown for illustration purposes only.

For example, vnode 118 can represent a file in a UFS file system. In this case, the vnode 118 contains a reference to an inode 106 that holds particular information on the file's representation on the data storage medium. An inode 106 is used to represent a Unix File System (UFS) file and is linked to the associated data storage medium 110 that stores the file. Further, vnode 118 can represent a NFS file system. In this case, the vnode 118 contains a reference to a mode 108 that represents a Network File System (NFS) file. The mode 108 is linked to a network interface 112 that is used to access the remote data storage medium containing the file. However, it should be noted that the present invention is not limited to UFS or NFS file systems and other types of file systems can be used in this context as well.

Alternatively, a vnode 117 can represent a remote file system resource where the file system specific data is a proxy object reference (PxFobj) that refers to vnode 118. From the viewpoint of the operating system running on node 102a, there is no difference between vnode 117 or any other vnode 118 existing on that node. The operating system accesses each vnode in the same manner.

The vnode 118 is linked to a VFS 120 that represents the overall file system. A VFS 120, 121 represents the overall file system. Each VFS 120, 121 contains a reference to data that is specific to the kind of file system that it represents (i.e., file system specific data). For example, VFS 120 represents a particular file system and contains a reference to file system specific data (not shown). Proxy VFS 121 represents a remote file system resource and its file system specific data is a proxy object reference (PxVFS) 124 that refers to FSobj 116. From the point of view of the operating system running on node 102a, there is no difference between proxy VFS 121 or any other VFS 120 existing on that node. The operating system accesses each VFS in the same manner.

It should be noted that from the client node's viewpoint, the proxy file system is just another file system type on par with an NFS or UFS file system. However, internally it operates by pairing proxy vnodes 117 on each client node with corresponding vnodes 118 on the server node. The file system specific data for the proxy file system (i.e., the PxFobj and the PxVFS) contains the linkage information required to maintain the association with the corresponding server node.

A more detailed description of the proxy file system can be found in Matena, et al., "Solaris MC File System Framework," Sun Microsystems Laboratories Technical Report SMLI TR-96-57, October 1996 which is hereby incorporated by reference as background information.

The server-side cluster layer of the file system includes a file object (Fobj) 114 representing the file, *myfile.c*. The Fobj 114 is linked to the vnode 118 associated with the file. In addition, there is a file system object (FSobj) 116 representing the file system as a whole.

The client-side local layer of the file system includes a proxy vnode 117 representing *myfile.c*. The proxy vnode 117 is linked to a VFS 120 representing the file system associated with *myfile.c*.

The proxy vnode 117 is linked to a proxy file object (PxFobj) 122 which contains a reference to the associated Fobj 114 for *myfile.c*. The PxFobj 122 is associated with the cluster layer of the file system. The client node 102a can access the file *myfile.c* through a RPC utilizing the object reference contained in PxFobj 122. In addition, the VFS 121 is linked to a PxVFS 124 which contains a reference to the file system object FSobj 116 representing the file system. The client node can access

the file system object through a RPC utilizing the object reference in PxVFS 124.

The aforementioned description details some of the data structures used to support the distributed file system in the cluster environment of the present invention. A global mount mechanism is provided that utilizes these data structures as well as additional procedures and data structures to maintain a global name space for each node in the cluster. A brief synopsis of the global mount mechanism is shown in Figures 2A-2B and 3A-3B.

Figs. 2A - 2B illustrate a global name space 134 associated with each node 102 in the cluster. The global name space is a collection of file system resources, each containing a set of files where each file is represented by a pathname. The pathname can include one or more directories that are organized in a hierarchical structure. Each directory can serve as a mount point for incorporating a new file system resource into the global name space.

In this example, there is a file system 132 associated with a first node 102a that will be mounted into the global name space 134. At the completion of the global mount, the global name space 134 will appear as the single image shown in Fig. 2B. In order for the global name space to appear as a single image, the file system is mounted into the global name space at the same mount point in each node concurrently. In Fig. 2B, the common mount point is /mnt to which the file system 132 with root $directory_2$ is mounted.

Figs. 3A - 3B illustrate the changes made to the cluster layer of the file system in order to mount the additional file system with root $directory_2$ into the global name space. Fig. 3A shows the file system with respect to the mount point /mnt before the mount and Fig. 3B shows the file system with respect to the mount point /mnt after the mount.

As shown in Fig. 3A, each node 102 has a vnode 118 or proxy vnode 117 representing the mount point, which in this example is the directory /mnt. The vnode 118 or its proxy 117 are linked to the VFS associated with file system containing the mount point. For each client node 102a, 102c, the proxy vnode 117 for the mount point is linked to a proxy file object 122. For the server node 102b associated with the mount point, there is a file object Fobj 114 linked to the vnode 118. There is also a file system object 116 representing the file system which is linked to a corresponding VFS 120. This infrastructure is in place before the file system with root $directory_2$ is mounted into the global name space.

The file system with root $directory_2$ is mounted in the global name space at the mount point /mnt. The file system after the mount is shown in Fig. 3B. The proxy vnode 117 for the mount point /mnt is linked by a mounted_here pointer 181 to the VFS representing the file system containing root $directory_2$ The proxy VFS 151 is linked by a covered_vnode pointer 190 to the mount point vnode. In each client node, the VFS 151 is linked to a PxVFS 124. The server node includes a file system object FSobj 156 linked to a corresponding VFS 150.

In addition, the global mount mechanism permits a process to unmount a file system from the global name space. The unmount procedure is performed such that the file system is unmounted from the same mount point and at the same time from each node in the cluster. The unmount of the file system having root $directory_2$ shown in Fig. 3B will result in the file system representing the global name space shown in Fig. 3A.

The aforementioned overview has presented the infrastructure of the distributed file system and the global mount mechanism. A more detailed description of the global mount mechanism and its operation is presented below.

System Architecture

Fig. 4 illustrates the distributed computing system 100 embodying the present invention. A cluster of nodes 102 is interconnected via a communications link 104. Each node does not share memory with the other nodes of the cluster. The communications link 104 generically refers to any type of wire or wireless link between computers, such as but not limited to a local area network, a wide area network, or a combination of networks. The client/server computers use the communications link 104 to communicate with each other.

Each of the nodes 102 contains a number of data structures and procedures used to support the distributed file system and the global mount mechanism. Each node includes an operating system or kernel 160. In a preferred embodiment , the operating system 160 is the Solaris MC operating system, which is a product of Sun Microsystems, Inc. Background information on the Solaris MC operating system can be found in "Solaris MC: A Multi-Computer OS," Technical Report SMLI TR-95-48, November 1995, Sun Microsystems, which is hereby incorporated by reference.

The Solaris MC operating system is a UNIX based operating system. As such, in describing the present technology, UNIX terminology and concepts are frequently used in describing the present invention. However, this is for illustration purposes and is not to be construed as limiting the invention to this particular operating system or file system design.

In addition, each node 102 can contain the following:

- an operating system 160;
- one or more file system (FS) factory procedures 166 that are used to instantiate a file system or VFS on an invoking node;
- a VFS table 168 that stores one or more VFS 120, 121, 150, 151 data structures;
- a PxFobj table 169 that stores one or more PxFobj data structures;
- a vnode table 170 that stores one or more vnode

117, 118 data structures;

- a proxy VFS (PxVFS) table 171 that stores one or more PxVFS data structures;
- a file object (Fobj) table 172 that stores one or more file objects (Fobj);
- a file system object (FSobj) table 173 that stores one or more file system objects (Fobj);
- a file system resource configuration database 174 that stores all the file system (FS) factory procedures 166 within the cluster. The database 174 is accessed using a key including the file system type and file system resource that retrieves the FS factory procedure 166 corresponding to the requested resource;
- one or more cache objects 176. Each cache object 176 is associated with a PxFobj and a server-side provider object 177. A method associated with the cache object 176 is used to access a particular global lock 182;
- one or more provider objects 177. Each provider object 177 is associated with a Fobj and a client-side cache object 176. A method associated with the provider object 177 is used to enable read or write access to a particular vnode's global lock 182;
- a mount list 178 that records the file systems (VFS) within the cluster;
- one or more local mount cache objects 165 that are used to provide a local vnode with a global locking capability;
- a proxy local mount object table 167 that stores one or more proxy local mount objects;
- a VFS list client object 179 that interacts with the VFS list server object 164;
- a global mount procedure 175 that is used to mount and unmount resources;
- a bootstrap procedure 159 that is used to provide a local vnode with a global locking capability;
- as well as other data structures and procedures.

One of the nodes is designated a list server node 102s since it tracks information regarding the globally mounted file systems in the cluster. In addition to the above mentioned data structures and procedures, the list server node 102s stores a global mount list 162 delineating all the globally mounted file systems in the cluster and stores a VFS list server object 164 that maintains the global mount list 162 and generates the requisite infrastructure needed to support a mounted resource in the global name space.

Fig. 5A details the components of a vnode 118 and proxy vnode 117. A vnode 118 or proxy vnode 117 is used to represent each file and directory in the distributed file system. Each proxy vnode 117 or vnode 118 can include:

- a pointer 180 to a VFS representing the file system associated with the file or directory the vnode or proxy vnode represents;
- a mounted_here pointer 181 linking the vnode or proxy vnode to a VFS representing a file system that uses the vnode or proxy vnode as its mount point;
- a local lock 183 that allows the vnode or proxy vnode to be locked by processes local to the node;
- a method array pointer 184 that points to one or more methods used to perform operations on the vnode 118 or proxy vnode 117. An example of one such method is the lookup method that is used to find or generate a vnode;
- a data pointer 185 that points to file system specific data pertinent to the file the vnode represents. When the vnode is a proxy, the file system specific data is a PxFobj which includes a pointer to the associated Fobj;
- a flags array 186 including a proxy flag 187 indicating whether or not the vnode is a proxy and a global locking flag 188 indicating whether or not the vnode 118 that is otherwise part of the local name space should use the global locking facilities when it is locked or unlocked;
- as well as other data.

Fig. 5B details the components of a VFS 120, 121. A VFS 120, 121 is used to represent each file system. Each VFS can include:

- a covered_vnode pointer 190 that points to the vnode that is the global mount point for the file system associated with the VFS;
- a data pointer 192 that points to file system specific data. When the VFS is a proxy, this file system specific data is a PxVFS object, which in turn contains a reference to the file system object on the server;
- as well as other data.

Fig. 5C details the components of the PxFobj 122 which can include a global lock 182 as well as other data. The global lock 182 is used to perform atomic operations on a vnode.

The system architecture including the data structures and procedures used to support the global mount mechanism has been described above. Attention now turns to the operation of the global mount mechanism. There are two central aspects to the global mount mechanism. The first is the manner in which the global mount mechanism is used to mount a new file system into an existing global name space. The second is the mechanism for establishing the global name space initially. The operation of the global mount mechanism in an existing global name space is described first, followed by a description of the manner in which the global name space is generated initially.

Mounting and Unmounting a File System in the Global Name Space

The global mount mechanism mounts a file system

into the global name space at a common mount point on each node of the cluster. In addition, the mount occurs concurrently on each node in the cluster. A global lock is used to lock the proxy vnodes representing the mount point on all nodes while the mount mechanism is in operation. This ensures that no other process can alter the mount point while the global mount or unmount operation is proceeding.

Similarly, the global mount mechanism unmounts a file system from the global name space from a common mount point concurrently in each node in the cluster. The global lock is used to lock the vnode of the mount point when the unmount operation is in operation.

Furthermore, the global mount mechanism allows one node to mount a file system whose resources reside in another node. This can occur, for example, when the NFS protocol stack is constrained to run on a single designated node, or when a block special file whose media contains a UFS file system is usable only from the nodes where the hardware is connected. The global mount mechanism determines which node is appropriate to be the server for the resource and then utilizes the server's file system factory to instantiate the resource as a file system object. After the factory has instantiated the file system, the global mount mechanism uses the list server to add the file system to the list of globally mounted file systems as well as notify each client node of the new globally mounted file system. Each client node in turn will set up the requisite data structures needed to mount the file system at the mount point concurrently.

Fig. 6 illustrates the steps used to mount a file system in the global name space. A user associated with a process issues a global mount command at a requesting node (step 200). The global mount command can have the following syntax:

mount -g <-F file system type> <resource> <mount point> where the -g indicates that the resource is to be mounted into the global name space,

the *-F* indicates that the following argument is a *file system type* that can take one of many possible values; two commonly used values are:

ufs, indicating a Unix file system or

nfs, indicating a network file system,

the *resource* field indicates the file system resource that will be mounted, and

the *mount point* field indicates the mount point or location in the global name space where the resource is to be mounted.

Upon receiving a global mount command, the global mount procedure 175 executes a lookup method to find the proxy vnode 117 associated with the mount point in the requesting node (step 202). If no proxy vnode 117 exists for the mount point in the requesting node, the lookup method will generate a proxy vnode 117 for the mount point and link it to the VFS 121 representing its containing file system. The method will also determine the server for the mount point and request that the server generate a file object Fobj 114 for the mount point. In response to this request, the server will, if as yet nonexistent, generate the file object Fobj 114 and link it to the corresponding server-side vnode 118 representing the mount point. In addition, the server will generate a provider object 177 for the requesting node. An object reference to the Fobj 114 is then returned to the requesting node and is stored in the proxy file object PxFobj 122. The PxFobj 122 is then linked to the proxy vnode 117 of the mount point. Additionally, information is also returned to the requesting node for it to generate an associated cache object 176.

Once the mount point vnode 118 is generated on the server and its corresponding proxy 117 is generated in the requesting node, the global mount procedure 175 acquires the vnode's global lock 182 for write access (step 204). This is accomplished by using a distributed locking scheme that employs a single writer/ multiple readers protocol. The locking scheme allows the mount or unmount operation to be performed on the mount point proxy vnode 117 while simultaneously blocking conflicting operations on the same mount point vnode 117 on the other nodes. The locking scheme is based on the distributed locking scheme recited in the "Decorum File System Architectural Overview," Proceedings of the USENIX Summer Conference 1990, pgs. 151-163, which is hereby incorporated by reference as background information.

The object of the locking scheme is to ensure that only one process has write access to the vnode 117, 118 at a time or that multiple processes have concurrent read access to the vnode 117, 118 at a time. This scheme is implemented using cache 176 and provider 177 objects. The cache object 176 is used by the proxy file object PxFobj 122 to request the global lock 182. The global lock 182 can be acquired for either read or write access. The provider object 177 is used to coordinate the requested read or write access with the other nodes.

Fig. 7 illustrates the distributed locking scheme. All file access is performed through a proxy vnode 117. In order to perform an operation coherently on a vnode, the distributed locking scheme locks the collection of proxy vnodes 117 across all nodes to accomplish the coherent operation.

Each client node 102a, 102c has a PxFobj 122 associated with the file object 114 for the mount point. The PxFobj 122 has a cache object 176 that is used to request the file's global lock for either read or write access. The server 102b for the file has a provider object 177 for each client node. A provider object 177 is paired with a respective client-side cache object 176. The provider objects 177 are associated with the file object Fobj 114 associated with the vnode 118 for the mount point. A request for the vnode's global lock 182 is made using the PxFobj 122 to the cache provider 176. The request is transmitted from the cache provider 176 to the respec-

tive server-side provider 177 that coordinates with the other providers 177 to determine whether the access should be granted or should be blocked.

For a mount or unmount operation, write access is necessary. The request is made to the associated cache object 176 which in turn calls the corresponding provider 177. The provider 177 consults with all other providers 177 and determines whether or not the access can be granted.

The locking protocol allows a write access when no other read or write access is active. Alternatively, multiple read accesses can occur concurrently when there is no write access active. Thus, only one write access is allowed at a time. If another provider 177 has been granted either read or write access, an attempt will be made to invalidate the access. If this cannot be done, the requesting provider 177 will wait until the outstanding write access is completed before it is granted write access.

Referring back to Fig. 6, once the vnode's global lock 182 has been acquired for write access, the global mount procedure 175 determines the appropriate node that should become the server for the resource (step 206). At times the node servicing the global mount command may not be the appropriate server for the resource that will be mounted. This may be attributable to several different factors, such as constraints imposed by the resource. As noted above, disparity in resource distribution can occur.

In order to accommodate this problem, the global mount procedure 175 determines which node is best suited to act as the server for the resource (step 206). This is accomplished by querying the cluster file system resource database 174 for the appropriate file system factory (fs_factory) procedure 166. Each fs_factory procedure 166 is associated with a particular node and used to instantiate a file system and VFS. The global mount procedure 175 then invokes the appropriate fs_factory procedure 166 to generate a VFS 150 in the server node for the mounted resource and a corresponding file system object FSobj 156 (step 206).

Next, the global mount procedure 175 calls the list server 102s with information about the newly generated FSobj 156 and the resource used to instantiate it (step 206). The list server 102s adds the newly generated FSobj 156 to the cluster-wide global mount list 162 (step 206). In addition, the list server 102s contacts each node 102 in the cluster and informs it of the mounted resource and mount point and transmits an object reference to the corresponding FSobj 156 (step 206). Each node 102, in turn, searches for a proxy VFS 151 for the FSobj 156 and a proxy vnode 117 for the mount point Fobj 114 (step 206). If these proxies do not exist in the node, they are created. When the proxy vnode 117 for the mount point is created, the file system specific data or PxFobj 122 is generated as well. Similarly, when the proxy VFS 151 is created, the file system specific data or PxVFS 124 is generated as well. Then the node 102 updates the mount list 178 with the proxy VFS 151 (step 206).

The next step is to splice the resource into the global name space at the mount point (step 208). The global mount procedure 175 performs this task in each node by setting the mounted_here pointer 181 of the mount point's proxy vnode 117 to the proxy VFS 151 representing the mounted resource. The covered_vnode pointer 190 of the proxy VFS 151 representing the mounted resource is linked to the vnode 117 of the mount point. When the mounted_here pointer 181 is set, this indicates that a file system has been mounted at the mount point. Finally, the global lock of the mount point's vnode 117,118 is released (step 210).

Figures 8A - 8D illustrate the mount of the file system 132 shown in Fig. 2 into the global name space. Referring to Fig. 8A, a client node 102a receives a global mount command specifying that an NFS file system is to be mounted in the global name space at the mount point /mnt. The requesting node does not have a vnode 118 or proxy vnode 117 for the mount point. However, on server node 102b, the Fobj 114 already exists.

Fig. 8B illustrates the file system 100 after the client node 102a looks up the mount point (step 202). The lookup method generates a proxy vnode 117 for the mount point on the client node 102a. A reference to the Fobj 114 is transmitted to the client node 102a and stored in a newly created proxy file object PxFobj 122a.

Fig. 8C illustrates the file system 100 after the list server 102s (not shown) generates the necessary infrastructure to instantiate a file system from the designated file system resource (step 206). A VFS 150 is generated in the server node 102b for the file system associated with the mounted resource and a corresponding FSobj 156. The list server 102s passes an object reference to the FSobj 156, an object reference to the mount point object 114, and a copy of the arguments to the mount command to each client node. In turn, each client node 102 generates a proxy vnode 117 for the mount point as well as a proxy VFS 151 representing the mounted resource and file system specific data 124 for that proxy VFS 151.

Fig. 8D represents the file system after the mounted resource is spliced into the global name space (step 208). The proxy vnode 117 for the mount point has its mounted_here pointer 181 linked to the VFS 151 representing the new file system, and the VFS 151 has its covered_vnode pointer 190 set to the proxy vnode 117 representing the mount point.

Fig. 9 illustrates the steps used by the global mount mechanism 175 to unmount a mounted resource from the global name space. Fig. 10 illustrates an exemplary unmount of the mounted file system illustrated in Figs. 8A - 8D. A process associated with a node receives a global unmount command (step 212). The global unmount command can have the following syntax:

unmount <mount point> where the *mount point* field indicates the mount point or location in the global name space where the file system resource is to be un-

mounted from.

The global mount mechanism 175 will then look up the proxy vnode of the unmounted resource utilizing the same steps described above (step 214). In Fig. 10, the unmounted resource's root directory is represented by proxy vnode 236. After finding the unmounted resource's root vnode 236, the global mount mechanism 175 obtains the associated VFS 151 through the VFS pointer 180. The VFS's 151 covered_vnode pointer 190 is traversed to vnode 117, which represents the mount point. The mount point vnode 117 is then locked in accordance with the locking mechanism described above (step 216).

Once the mount point vnode 117 is locked, the list server 1 02s is called to have each node unsplice the file system resource from that node's mount point proxy vnode 117 (step 218). This is performed by deleting the contents of the mounted_here VFS pointer 181 (step 218).

The global mount mechanism 175 will then delete the infrastructure used to support the unmounted resource (step 220). The list server 102s will delete the VFS 121 from the global mount list 162 and inform the other client nodes to delete their VFS 121 and PxFobj 122 data structures as well.

The global lock 182 representing the mount point is then released in a similar manner as was described above (step 222).

The above description details the manner in which the global mount mechanism mounts or unmounts a file system to and from the global name space. Attention now turns to the manner in which the global name space is generated initially.

Generating the Global Name Space

A characteristic of the global name space is that each directory in the global name space can serve as a global mount point. New file systems can be incorporated into the global name space at a global mount point. A distinguishing feature of a global mount point is that it can be locked globally.

Initially, when each node in the cluster "boots up", the global name space does not exist. Instead, each node has a set of local vnodes representing the file resources associated with a particular node. The first step in incorporating these local vnodes into the global name space is to provide each local vnode with a global locking capability. The global locking capability allows a local vnode representing the same file resource in each node to be locked concurrently. Once the local vnode acquires a global locking capability, the local vnode can be used as a global mount point and a mount can be established there that becomes a part of the global name space.

In order for a local vnode to acquire the global locking capability, a distributed locking mechanism is generated to prevent two or more nodes from establishing the global lock capability for the same local vnode at the same time. Thus, only one node needs to perform the global namespace initialization procedure.

One node will request that a local vnode or mount point be granted global locking capability. The list server 102s acts as the server for the local vnode or mount point and generates a local mount object 189 to represent the mount point in the list server 102s. The list server 102s also generates a local mount provider object 161 for each node in the cluster. The list server 102s then visits each node in the cluster, providing the node with enough information for the node to construct a proxy local mount (pxlocalmnt) object 167 and a local mount cache object 165 in the client node. The cache/provider object pair is then used to lock the mount point that is distributed in each node in the cluster. As each node is visited, locking responsibility for the vnode representing the mount point on that node is transferred from the vnode itself to the cache/provider pair newly associated with that vnode.

Figs. 11 and 12 illustrate the steps used to provide a mount point with a global locking capability. These steps can be performed by an initialization procedure 159. A node 102 contacts the list server 102s with a request to provide a mount point with global locking capability (step 230). The list server 102s creates a local mount object 189 representing the mount point in the list server 102s and a local mount provider object 161 for each node in the cluster (step 232).

The list server 102s then "visits" each client node by calling the list client on each node in turn, starting with the client node 102a that initiated the request (step 234). The list server 102s provides information to each client node 102a, 102c which the client node's 102a, 102c list client uses to perform the following tasks. The client node 102a, 102c will perform the pathname lookup method on the mount point, thereby generating a vnode 118 for the mount point. The list server 102s will send the client node 102a, 102c an object reference to the local mount object 189 which the client node 102a, 102c uses to construct a proxy local mount object 167 (pxlocalmnt). The pxlocalmnt 167 is linked to the vnode 118 representing the mount point. In addition, the client node 102a, 102c generates a local mount cache object 165 that is paired to a corresponding local mount provider 161 (step 234).

Next, the local lock 183 associated with the mount point's vnode 118 is acquired. This is performed in a similar manner as was described above with respect to Figures 6 and 7 (step 234).

Once the local lock 183 is acquired, the global lock flag 188 in the mount point's vnode 118 is turned on. When the global lock flag 188 is turned on or set, this indicates that the associated mount point has acquired the global locking capability. The requesting client node 102a is then given write access to the global lock 182. Lastly, the local lock 183 is released (step 234).

This procedure is performed in each client node

102a, 102c (step 234). At the completion of this procedure, the mount point has acquired global locking capability.

It then can be used as a mount point for a new file system resource . In this case, the global mounting procedure 175 described above can be used to mount a new file system resource at the mount point.

Accordingly, the global name space is generated by creating global mount points from the local vnodes in each node's local namespace. New file system resources can then be mounted into the global name space at these newly created global mount points. The progression of these steps will generate the global name space.

Alternate Embodiments

While the present invention has been described with reference to a few specific embodiments, the description is illustrative of the invention and is not to be construed as limiting the invention. Various modifications may occur to those skilled in the art without departing from the scope of the invention.

The present invention is not limited to the computer system described in reference to Fig. 1. It may be practiced without the specific details and may be implemented in various configurations, or makes or models of distributed computing systems, tightly-coupled processors or in various configurations of loosely-coupled microprocessor systems.

Further, the method and system described hereinabove is amenable for execution on various types of executable mediums other than a memory device such as a random access memory. Other types of executable mediums can be used, such as but not limited to, a computer readable storage medium which can be any memory device, compact disc, or floppy disk.

Claims

1. A method for maintaining a global name space in a computing system having a plurality of nodes interconnected by a communications link, the global name space representing a plurality of global file system resources accessible from each node, each global file system resource including a plurality of file resources, the global name space including global pathnames with each global pathname representing one of the global file resources, each global pathname including one or more global directories, wherein the global name space is distributed over the nodes;

   the method comprising the steps of:

   (a) providing a first file system resource for mounting in the global name space at a designated first mount point selected from the global directories;

   (b) concurrently locking the first mount point in each node;

   (c) mounting the first file system resource at the first mount point in each node; and

   (d) concurrently unlocking the first mount point in each node.

2. The method of claim 1, comprising:

   providing a second file system resource for unmounting from the global name space at a second mount point selected from the global directories;

   concurrently locking the second mount point in each node;

   unmounting the second file system resource from the second mount point in each node; and

   unlocking the second mount point in each node.

3. The method of claim 1 or claim 2, wherein:

   the computing system includes a plurality of local name spaces, each local name space representing local file system resources associated with one of the nodes, each local file system resource representing local file resources, the local name space including local pathnames, each local pathname representing one of the local file resources and including one or more local directories;

   the method including:

   enabling one or more of the local directories with a global locking capability that enables the enabled local directory to be locked by each of the nodes, the enabled local directory being a global directory in the global name space; and

   concurrently mounting one or more of the local file system resources into the global name space at a select one of the global directories, each mounted local file system resource being a global file system resource.

4. The method of claim 3, wherein the concurrently mounting step includes the steps of:

   (i) concurrently locking the select global directory in each node;

   (ii) mounting a first local file system resource at the select global directory in each node; and

   (iii) concurrently unlocking the select global directory in each node.

5. A computer system for maintaining a global name space, the system including a plurality of nodes interconnected by a communications link, the system

comprising:

a plurality of global file system resources accessible from each of the nodes, each global file system resource representing global file resources;

a global name space representing the global file system resources and including a plurality of global pathnames that each represent one of the global file resources, each global pathname including one or more global directories, each global directory being a global mount point to which a new file system resource can be mounted, the global name space distributed over the nodes;

a first locking mechanism, distributed over the nodes, for concurrently locking a same global directory in each node; and

a global mount mechanism, distributed over the nodes, for mounting a new file system resource into the global name space at a specified mount point in each node.

6. The system of claim 5, comprising:

a plurality of vnode mechanisms, each vnode mechanism representing in each node a global directory associated with the global name space;

a plurality of virtual file system (VFS) mechanisms, each VFS mechanism representing in each node a file system resource associated with the global name space; and

the global mount mechanism establishing a VFS mechanism for a newly mounted file system resource in each node, establishing a vnode mechanism in each node for a global mount point, and linking the vnode mechanism in each node to the VFS mechanism representing the new file system resource.

7. The system of claim 6, comprising:

a plurality of file system objects, each file system object (FSobj) representing a file system resource in a select node;

a plurality of proxy file system objects, each proxy file system object (PxFSobj) used to reference a corresponding FSobj; and

the global mount mechanism designating one of the nodes as a server node for a newly mounted file system resource, the server node generating a FSobj for the newly mounted file system resource, linking the FSobj to a corresponding VFS representing the newly mounted file system resource and passing an object reference to the newly mounted file system resource to all other nodes, each node receiving the object reference generating a proxy file system object (PxFSobj) from the object reference and linking the PxFSobj to a corresponding VFS representing the newly mounted file system resource in the respective node.

8. The system of any one of claims 5 to 7, comprising:

the global mount mechanism having means for unmounting a mounted file system resource from the global name space, the global mount mechanism locating a same mount point in each node from which the mounted file system resource is unmounted, concurrently locking the same mount point in each node, unmounting the mounted file system resource from the same mount point in each node, and unlocking the mount point in each node.

9. The system of any one of claims 5 to 8, comprising:

a plurality of local name spaces, each local name space associated with a particular node and representing local file system resources associated with the particular node, each local file system resource representing local file resources, the local name space including local pathnames, each local pathname representing one of the local file resources and including one or more local directories; and

an initialization mechanism that enables a first local directory with a global locking capability, thereby making the enabled local directory a global mount point, and that mounts a local file system resource into the global name space at a global mount point.

10. The system of claim 9, further comprising:

a second locking mechanism distributed in each node, the second locking mechanism having a capability to concurrently lock a same local directory in each node; and

the initialization mechanism using the second locking mechanism to lock the first local directory in each node to enable the first local directory with the global locking capability.

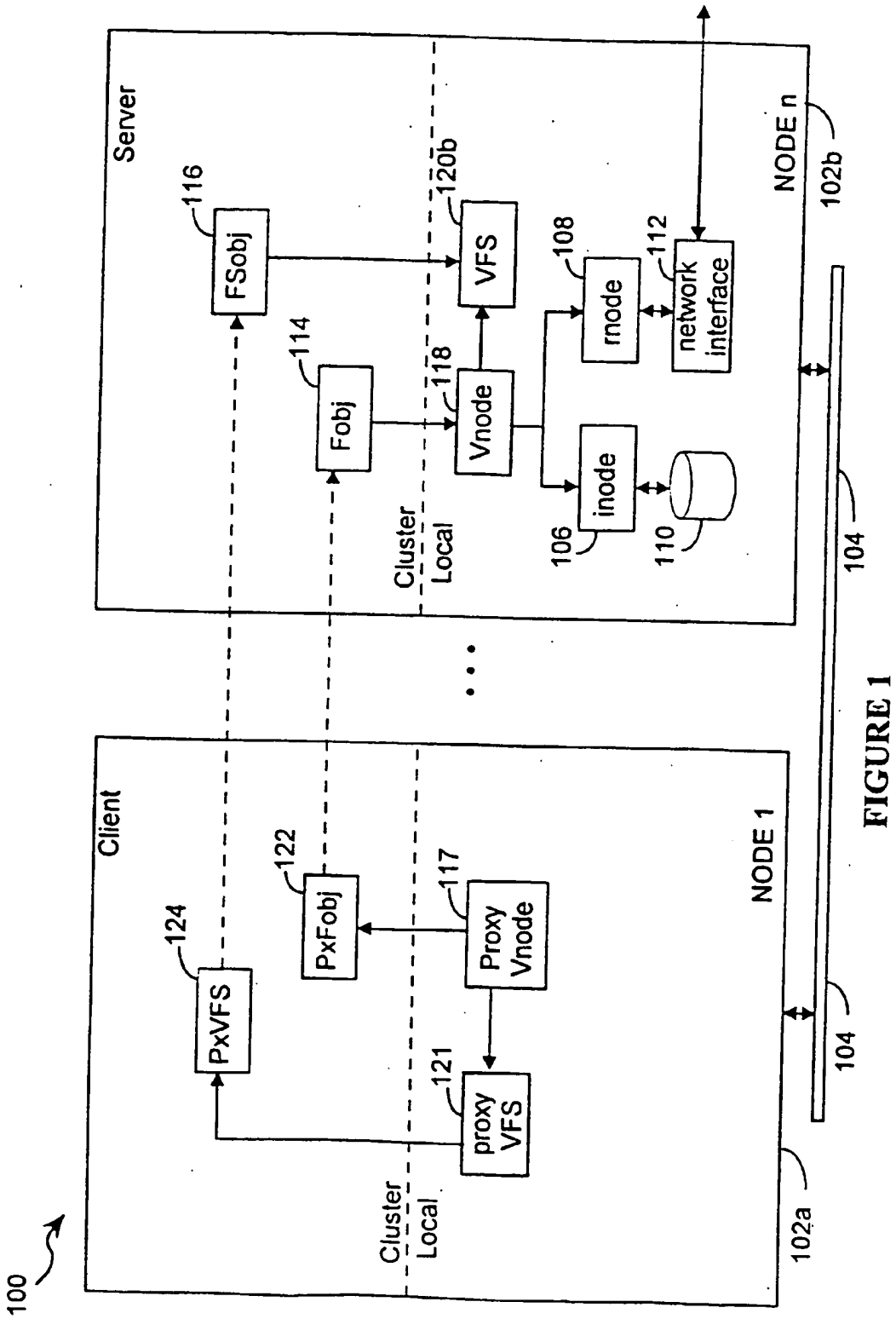

FIGURE 1

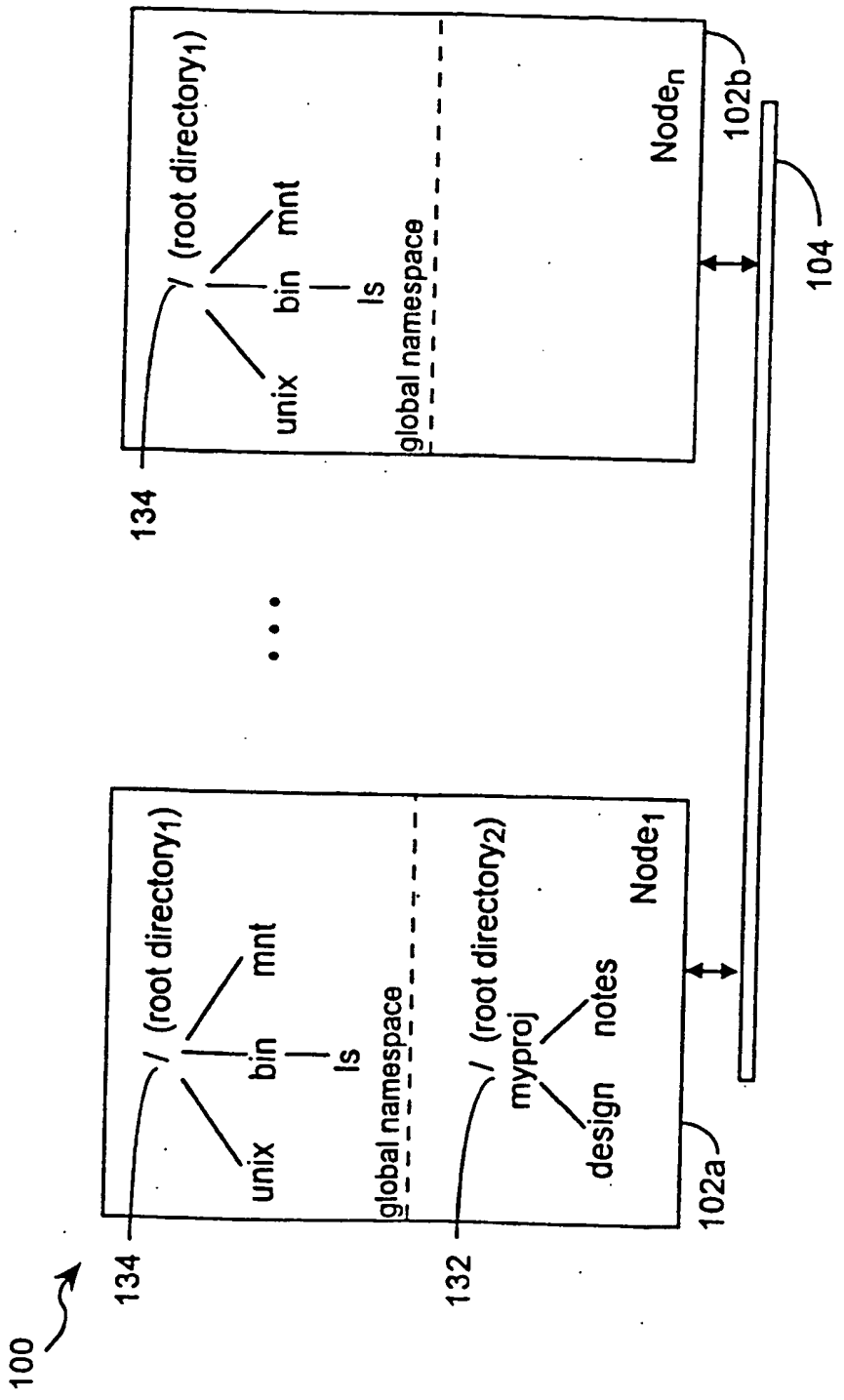

FIGURE 2A

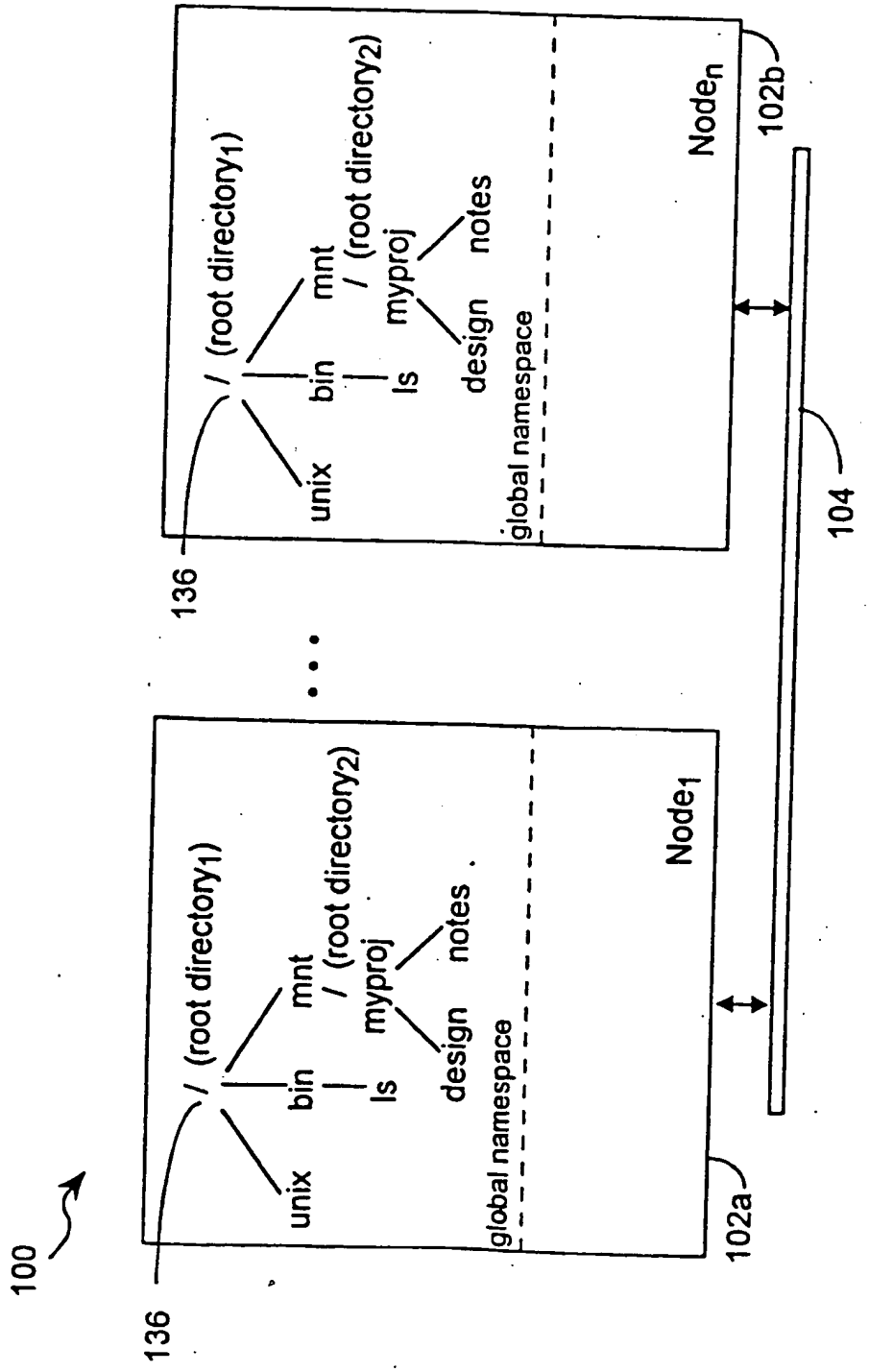

FIGURE 2B

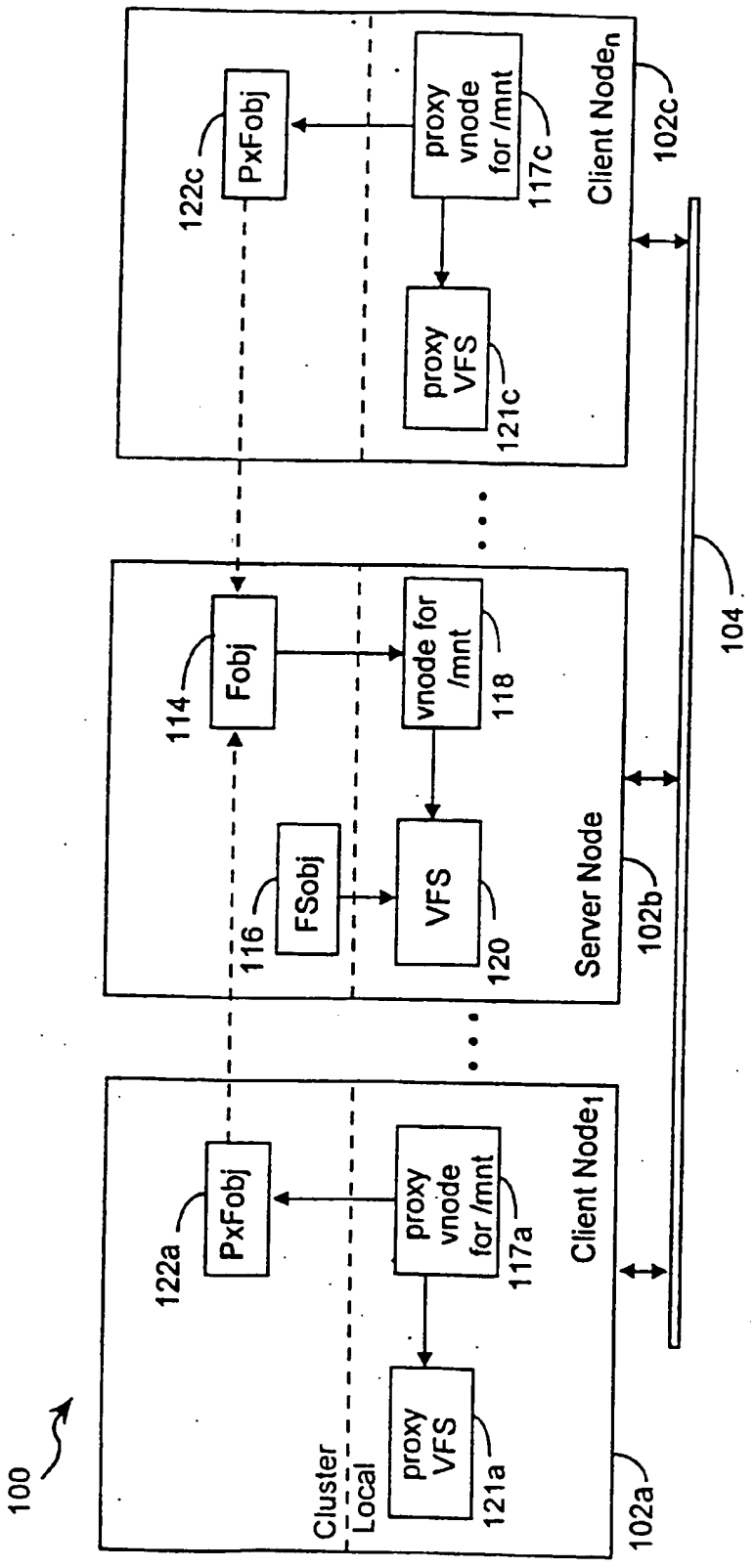

FIGURE 3A

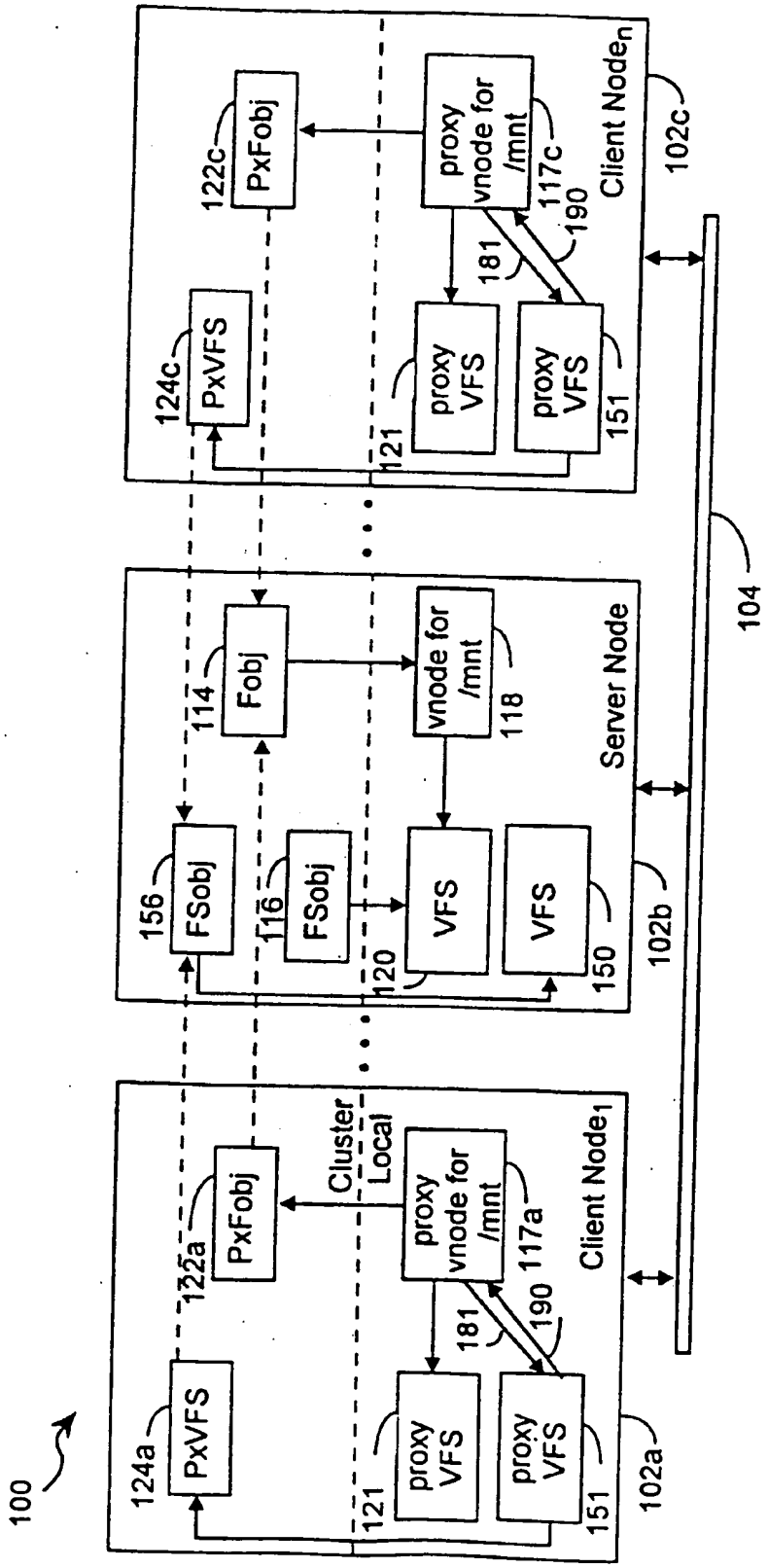

FIGURE 3B

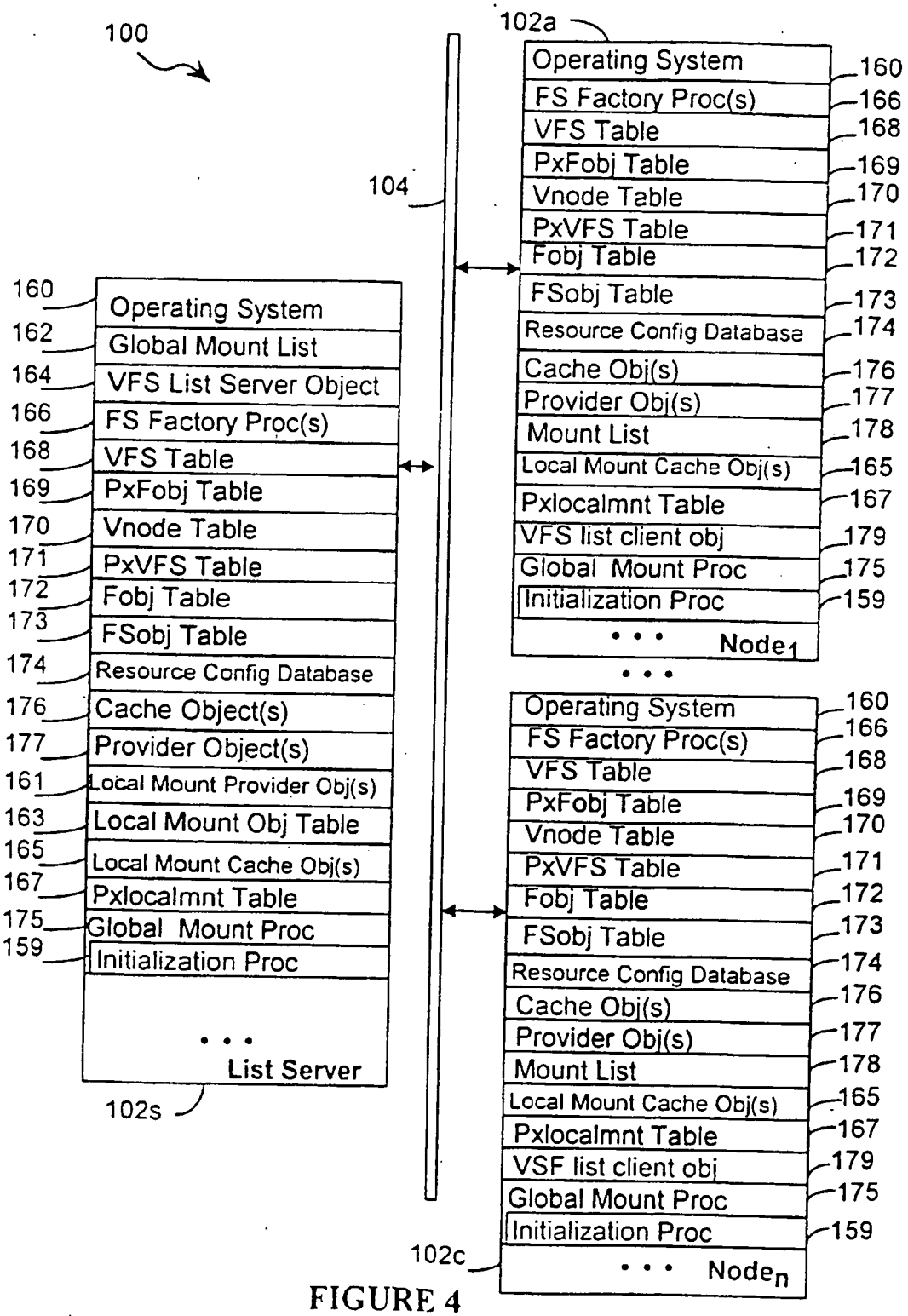

FIGURE 4

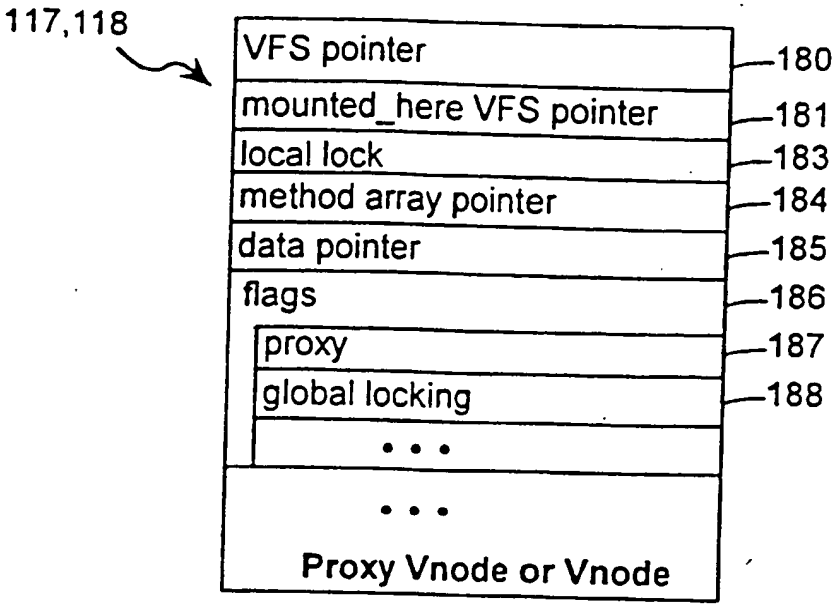

FIGURE 5A

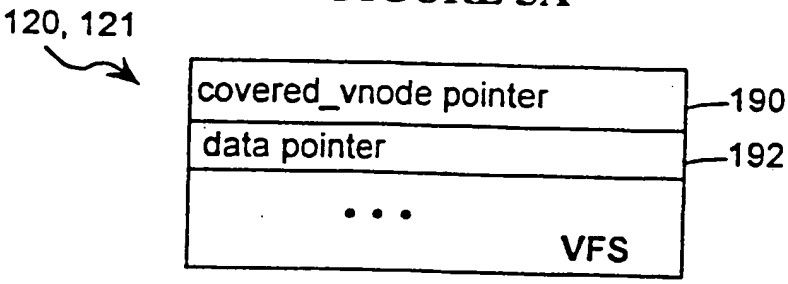

FIGURE 5B

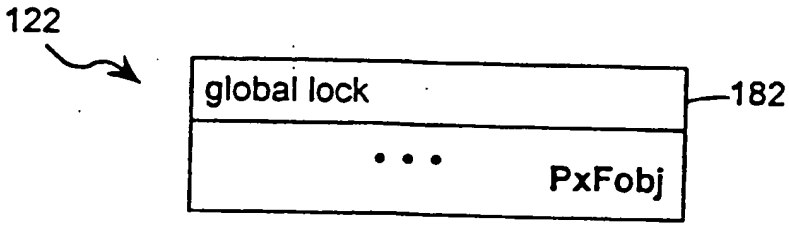

FIGURE 5C

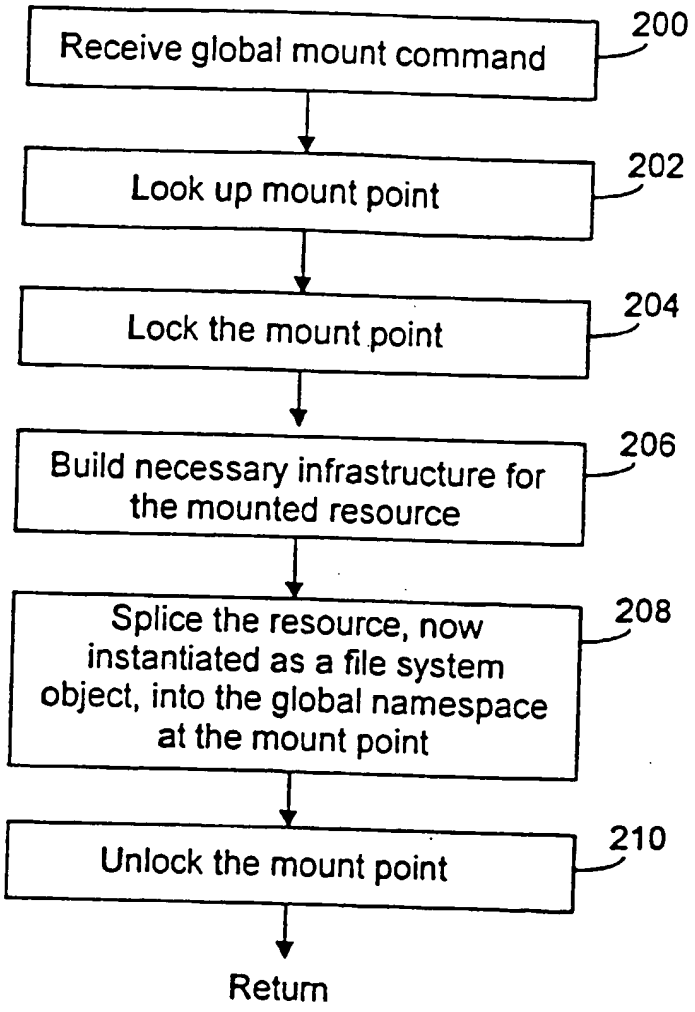

FIGURE 6

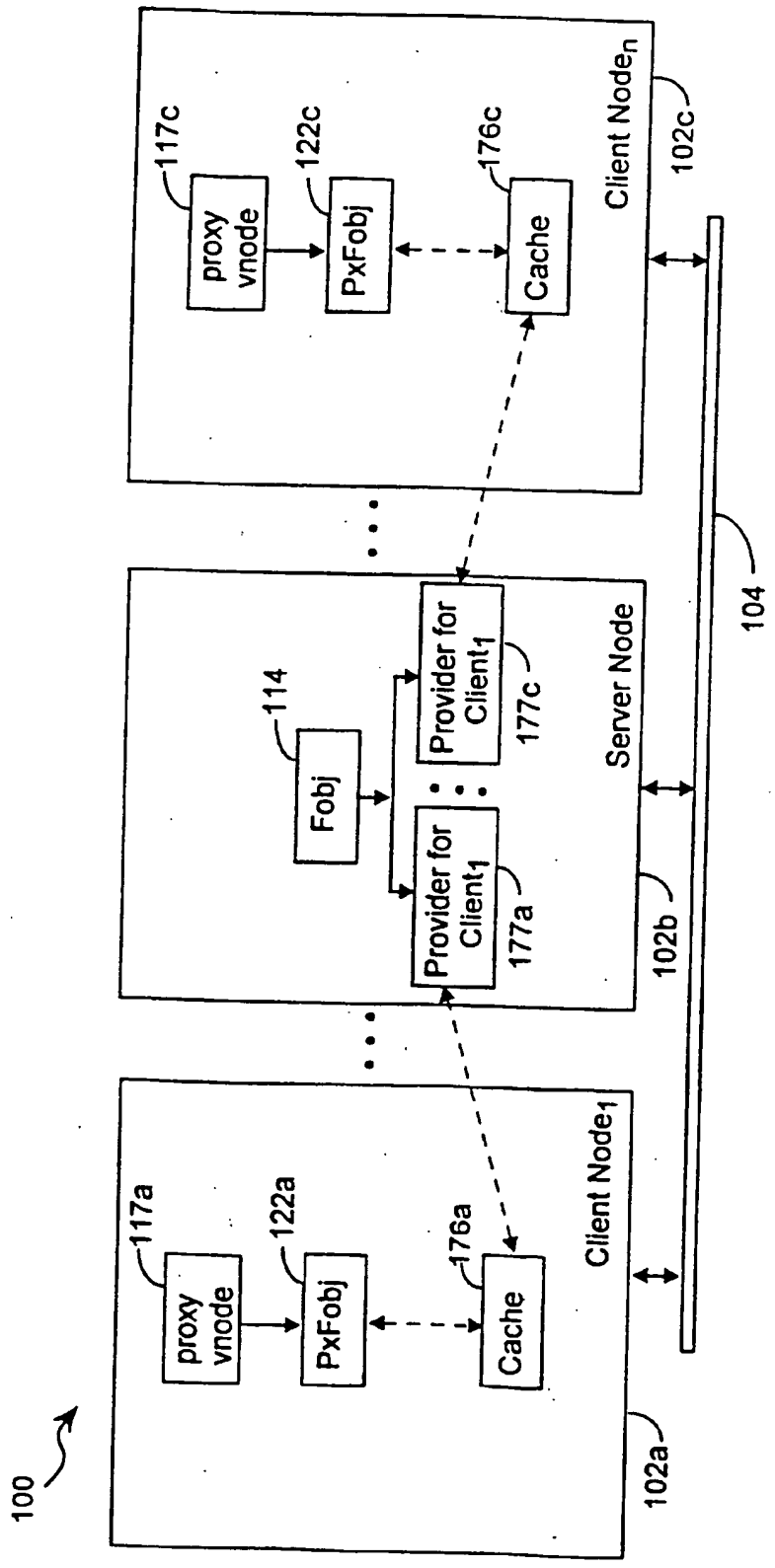

FIGURE 7

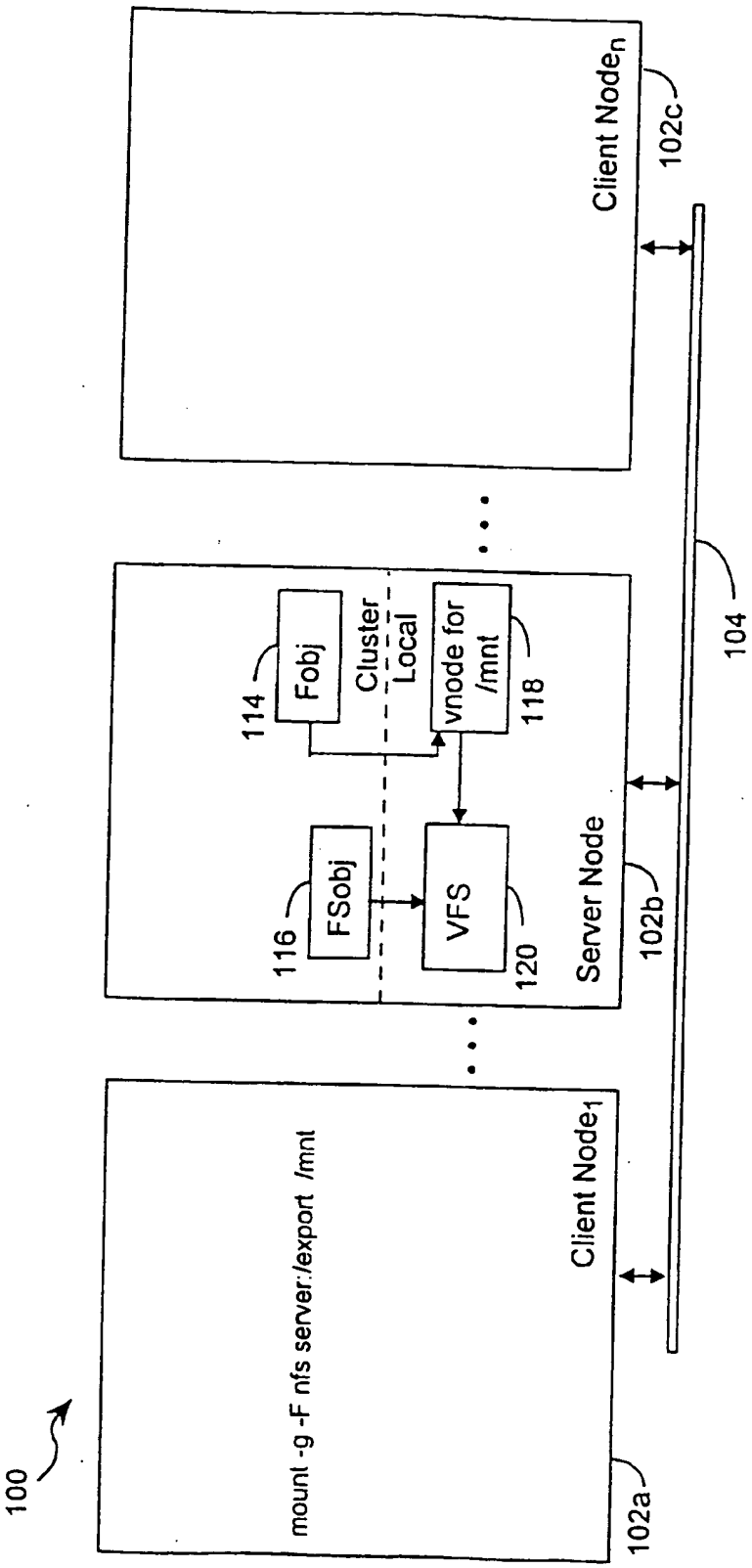

FIGURE 8A

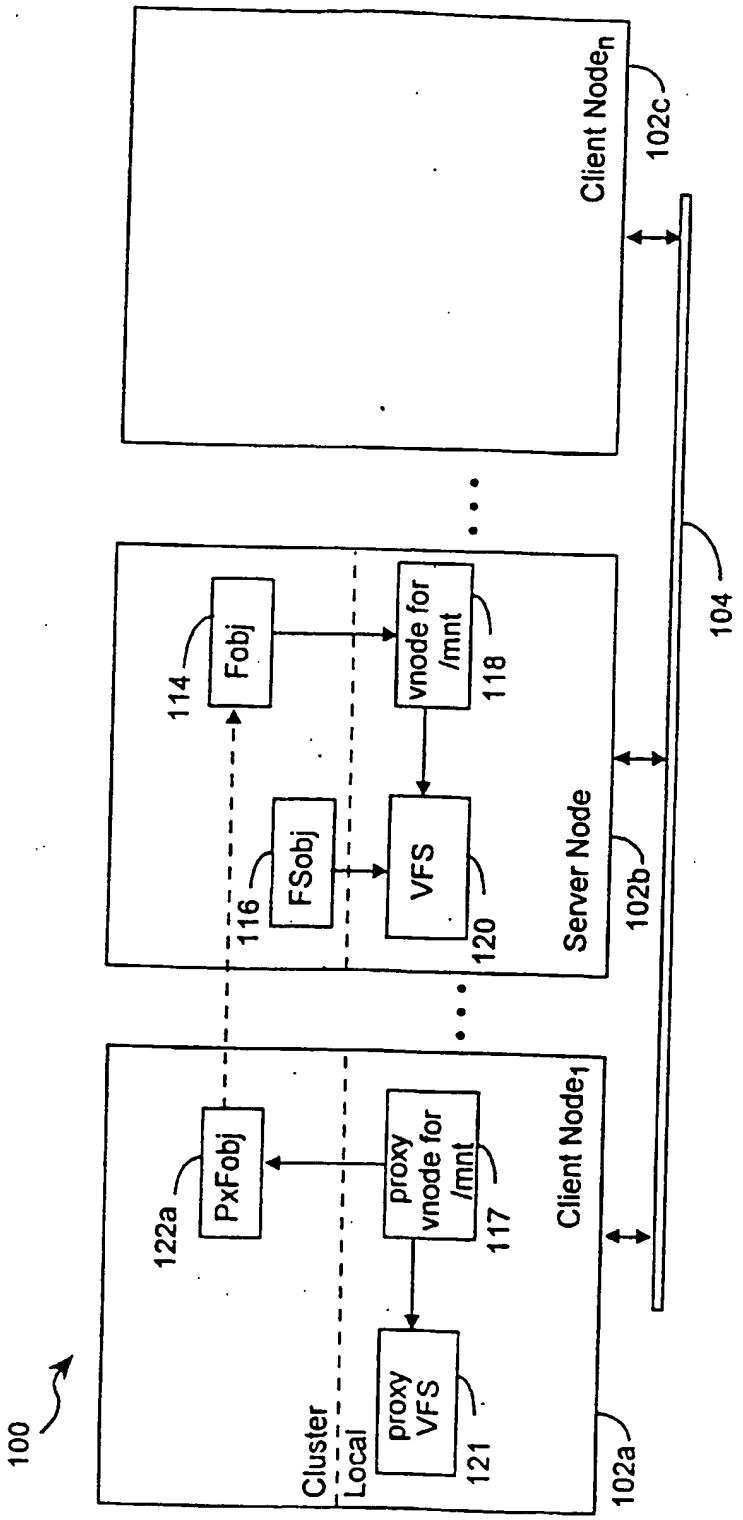

FIGURE 8B

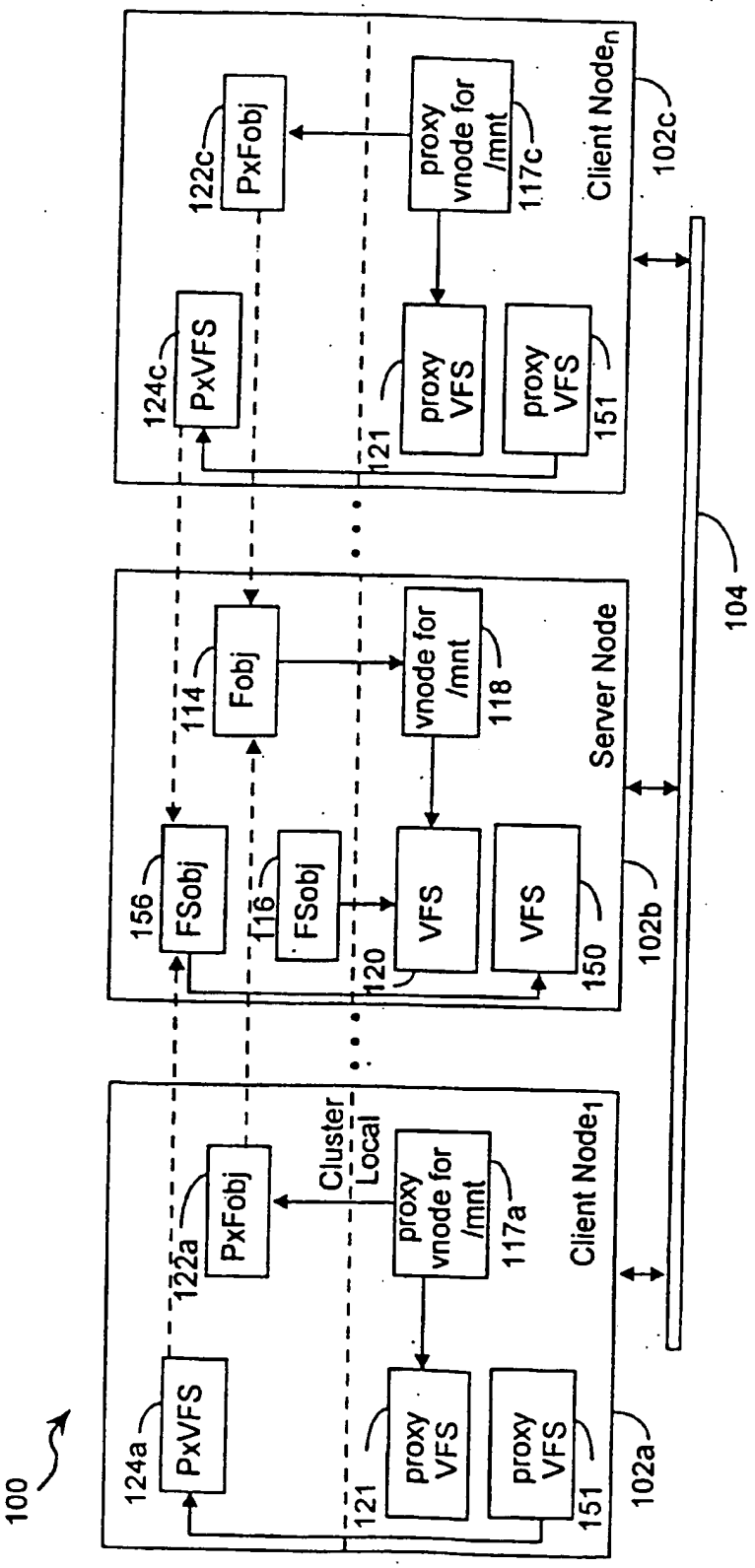

FIGURE 8C

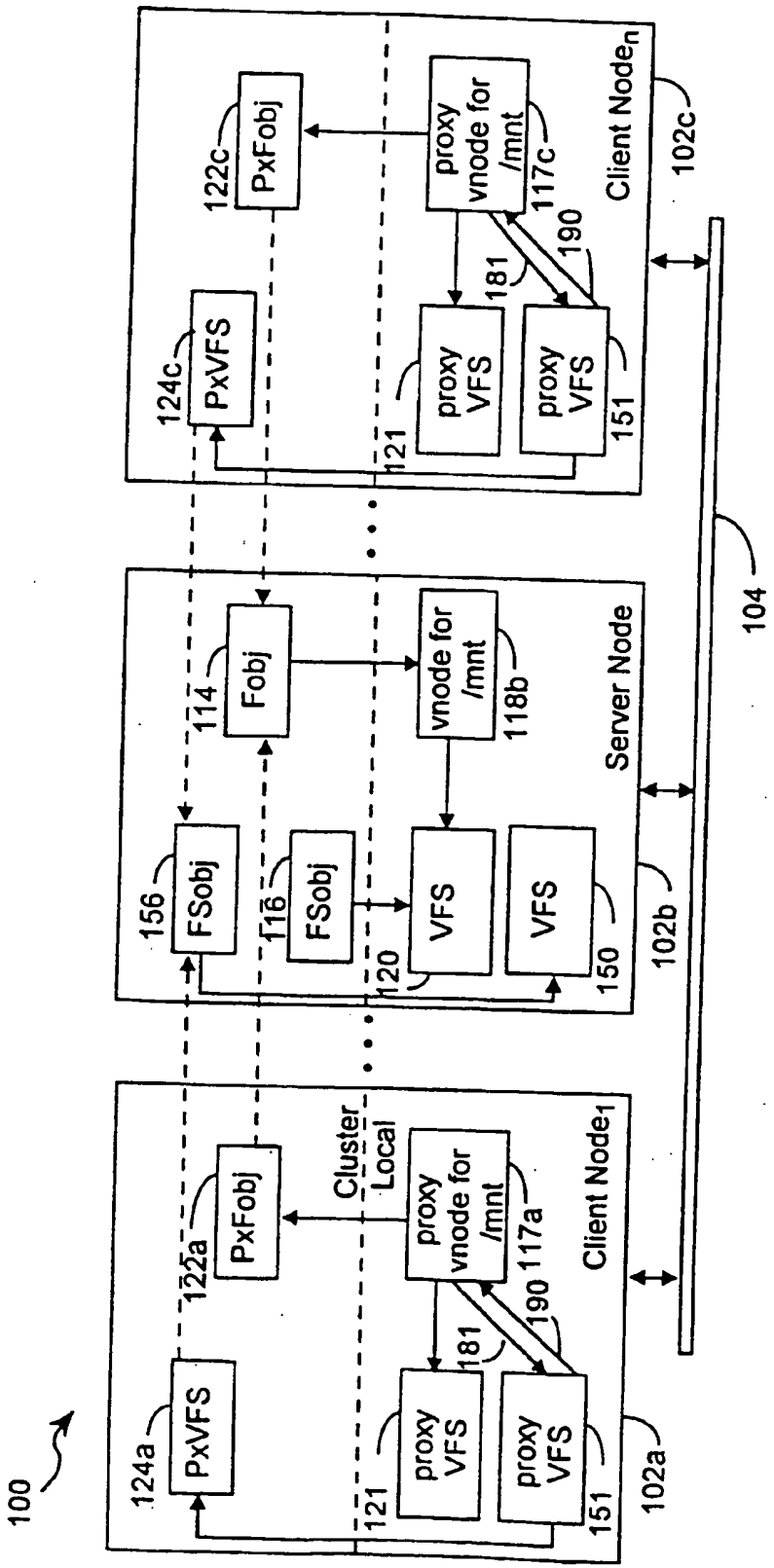

FIGURE 8D

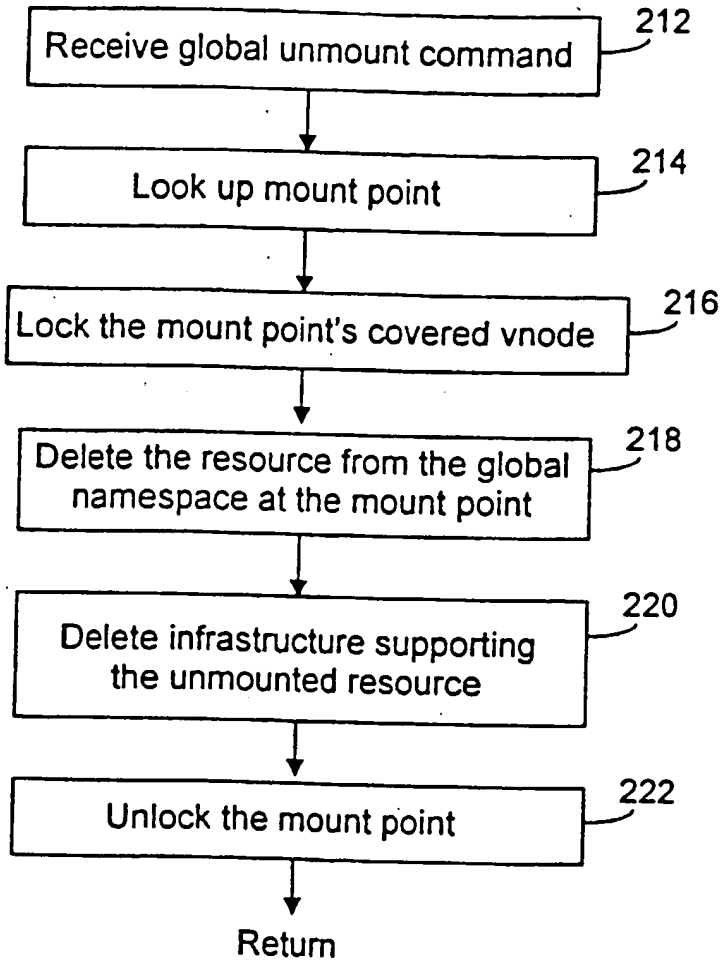

FIGURE 9

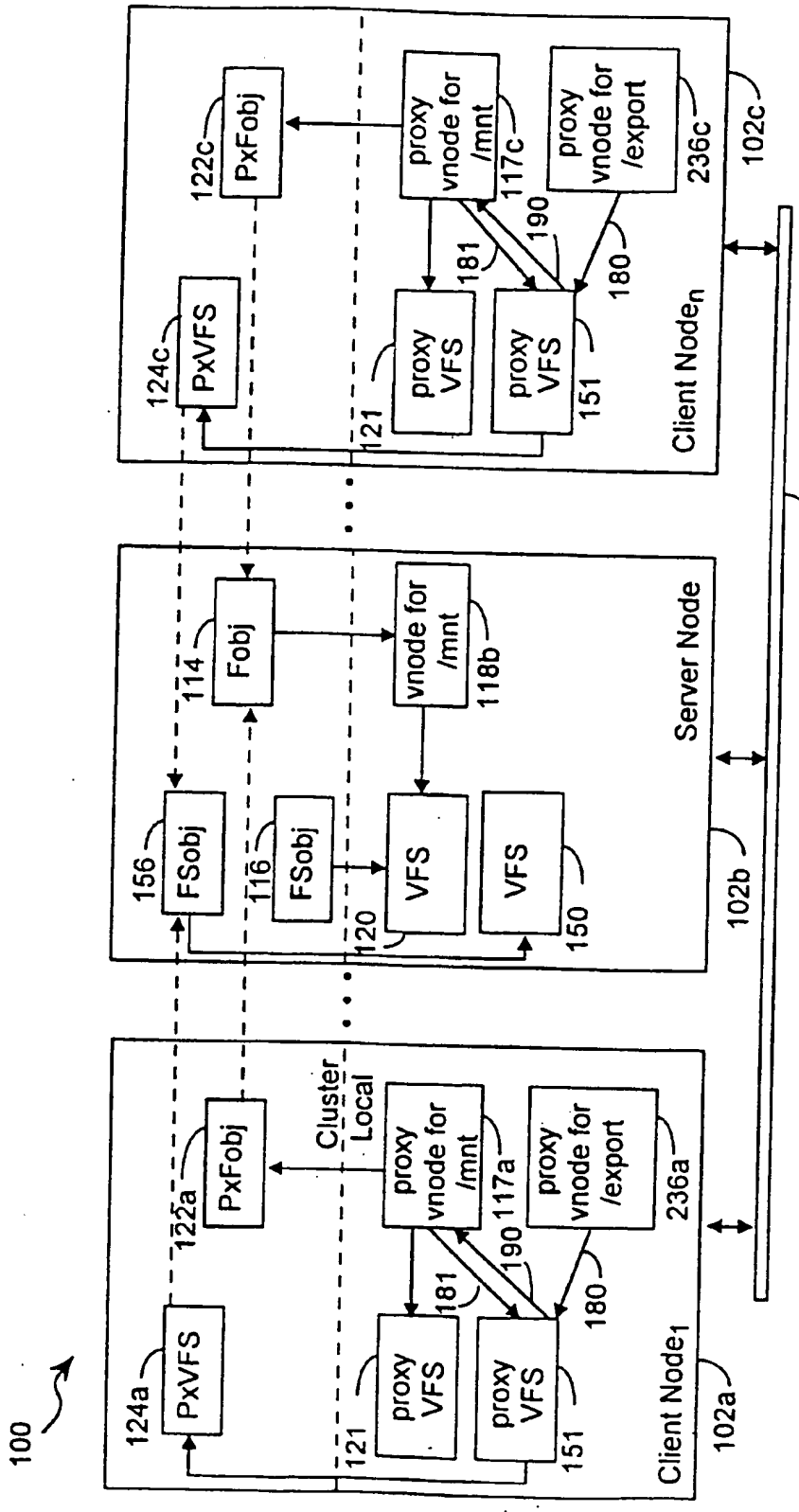

FIGURE 10

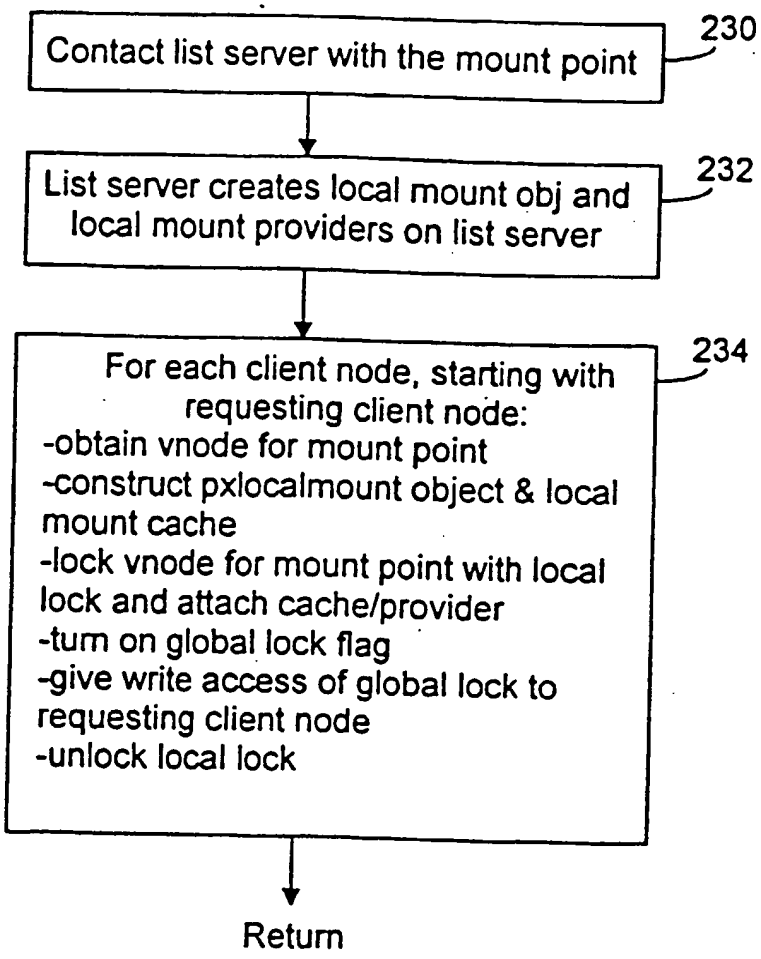

FIGURE 11

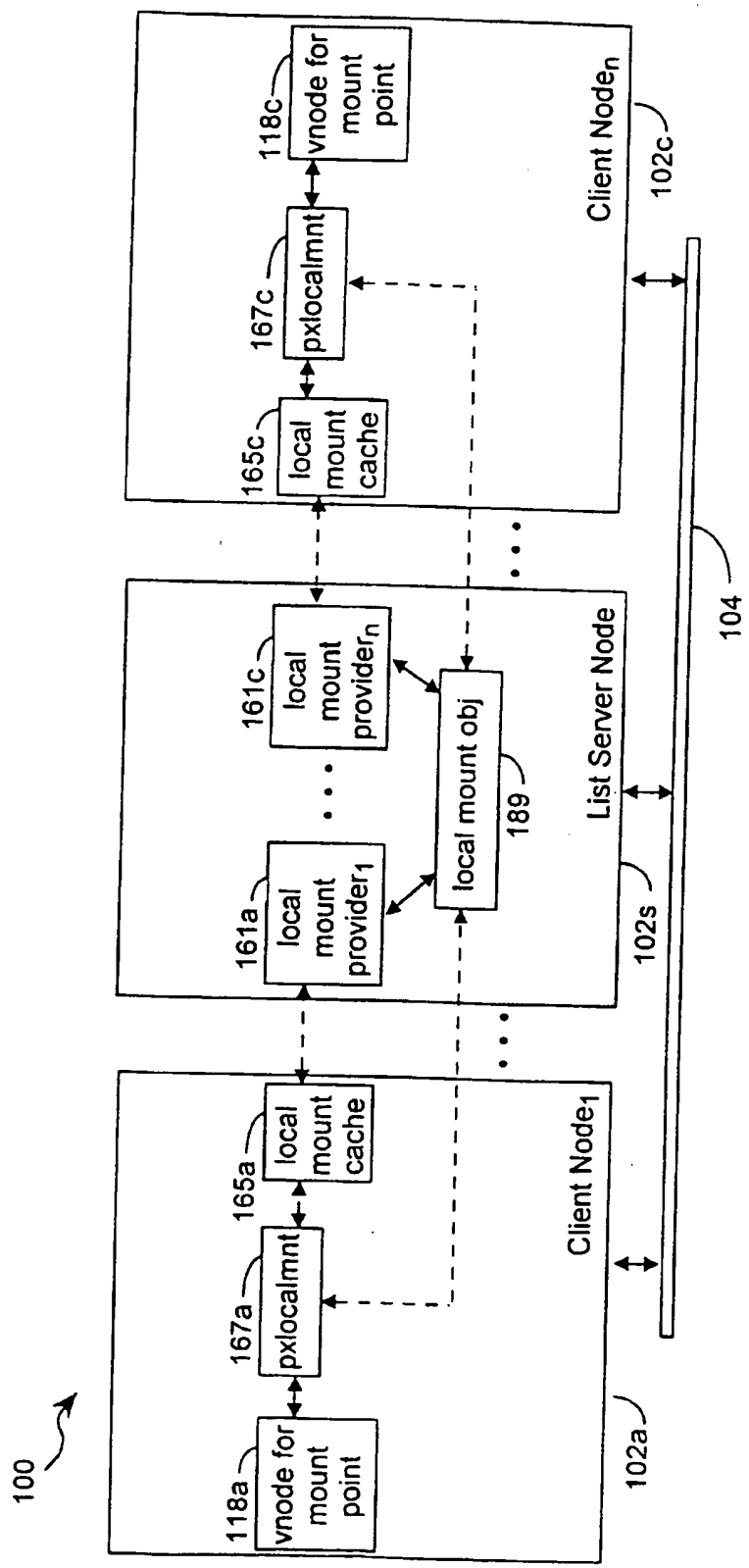

FIGURE 12

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 887 751 A3

(12) EUROPEAN PATENT APPLICATION

(88) Date of publication A3:
14.01.2004 Bulletin 2004/03

(51) Int Cl.7: **G06F 17/30**

(43) Date of publication A2:
30.12.1998 Bulletin 1998/53

(21) Application number: **98305007.1**

(22) Date of filing: **25.06.1998**

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **27.06.1997 US 884252**

(71) Applicant: **Sun Microsystems, Inc.**
Santa Clara, California 95054 (US)

(72) Inventors:
- **Cramer, Samuel M.**
  Mountain View, California 94040 (US)
- **Skinner, Glenn C.**
  Palo Alto, California 94306 (US)

(74) Representative: **Harris, Ian Richard et al**
D. Young & Co.,
21 New Fetter Lane
London EC4A 1DA (GB)

(54) Method and system for maintaining a global name space

(57) A global mount mechanism capable of maintaining a consistent global name space in a distributed computing system including a plurality of nodes interconnected by a communications link is herein disclosed. The global mount mechanism mounts a new file system resource into the global name space in a coherent manner such that the new file system resource is mounted at the same mount point concurrently in each node. The global mount mechanism accommodates mount or unmount requests initiated from a requesting node for a resource located in a remote node. The global mount mechanism is also used to unmount a file system resource from the global name space. The global mount mechanism also includes an initialization procedure that is used to generate the global name space initially by providing each local mount point with a global locking capability.

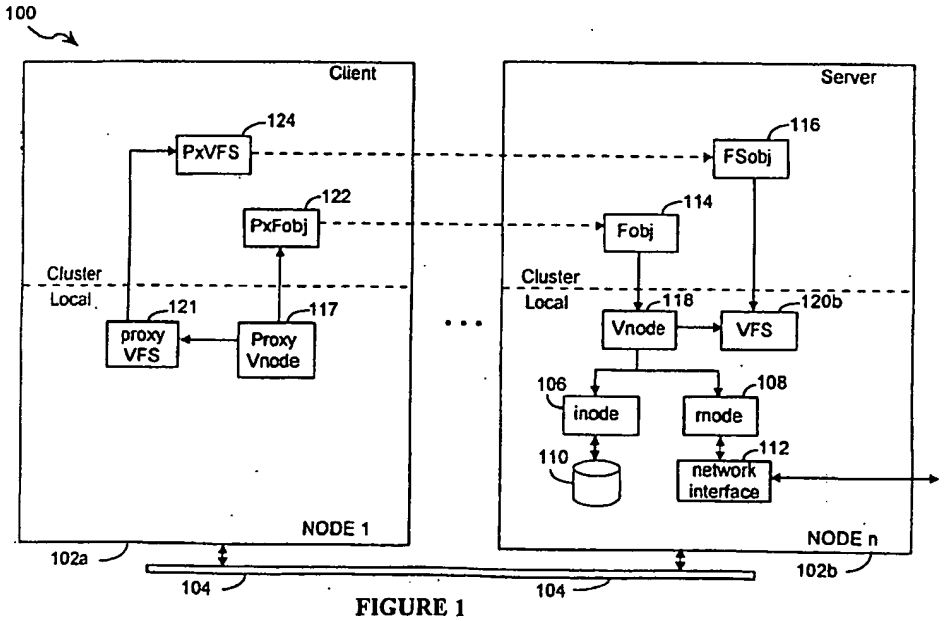

FIGURE 1

EP 0 887 751 A3

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 98 30 5007

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
X	"THE LOCUS DISTRIBUTED FILESYSTEM" LOCUS DISTRIBUTED SYSTEM ARCHITECTURE, MIT PRESS, 1985, pages 29-72, XP000748057	1,2,5,6, 8	G06F17/30
A	* page 36, line 22-26 * * page 38, line 17 - page 40, line 22 * * page 42, line 9-19 *	3,4,7,9, 10	
A	SATYANARAYANAN M: "A survey of distributed file systems" RESEARCH REPORT, February 1989 (1989-02), page 0-27 XP002260895 Department of CS, Carnegie Mellon University, USA * page 7, line 4 - page 14, line 20 * & SATYANARAYANAN M: "A survey of distributed file systems" ANNUAL REVIEW OF COMPUTER SCIENCE. VOL.4, 1990, PALO ALTO, CA, USA, ANNUAL REVIEWS, USA, pages 73-104, ISBN: 0-8243-3204-0	1-10	TECHNICAL FIELDS SEARCHED (Int.Cl.6) G06F
A	WELCH B: "A comparison of the Vnode and Sprite file system architectures" PROCEEDINGS OF THE USENIX FILE SYSTEMS WORKSHOP, ANN ARBOR, MI, USA, 21-22 MAY 1992, pages 29-44, XP001172859 1992, Berkeley, CA, USA, USENIX Assoc, USA Section 4	1-10	
A	EP 0 661 652 A (MICROSOFT CORP) 5 July 1995 (1995-07-05) * column 14, line 12 - column 18, line 1 *	1-10	

-/--

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	10 November 2003	Bykowski, A

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 98 30 5007

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
A	US 5 623 666 A (PIKE ROBERT C ET AL) 22 April 1997 (1997-04-22) * column 21, line 8 - column 31, line 34 *	1-10	

TECHNICAL FIELDS SEARCHED (Int.Cl.6)

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	10 November 2003	Bykowski, A

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

ANNEX TO THE EUROPEAN SEARCH REPORT ON EUROPEAN PATENT APPLICATION NO. EP 98 30 5007

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report. The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

10-11-2003

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0661652 A	05-07-1995	US 5701462 A	23-12-1997
		CA 2138630 A1	30-06-1995
		DE 69427681 D1	16-08-2001
		DE 69427681 T2	31-10-2001
		EP 0661652 A1	05-07-1995
		JP 7210443 A	11-08-1995
		US 5842214 A	24-11-1998
US 5623666 A	22-04-1997	AU 649455 B2	26-05-1994
		AU 8033491 A	16-01-1992
		CA 2046723 A1	12-01-1992
		DE 69129479 D1	02-07-1998
		DE 69129479 T2	05-11-1998
		EP 0466486 A2	15-01-1992
		ES 2116276 T3	16-07-1998
		JP 2724256 B2	09-03-1998
		JP 4233654 A	21-08-1992
		JP 10124376 A	15-05-1998
		CA 2045799 A1	12-01-1992
		DE 69130312 D1	12-11-1998
		DE 69130312 T2	22-04-1999
		EP 0466389 A2	15-01-1992
		JP 2719464 B2	25-02-1998
		JP 4233639 A	21-08-1992
		US 5457796 A	10-10-1995

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82